Amendments to the Specification

Please replace the paragraph beginning on page 4, line 6, with the following rewritten paragraph:

Multiple core chips are the result of ever increasing chip real estate due to shrinking circuit size. It is equivalent to shrinking a multiple processor SMP server onto a single piece of silicon. For example, Sun Microsystems™ plans to have a single chip with 8 cores, with each core capable of executing 4 threads simultaneously. This is the equivalent of a 32-processor machine on 1 chip. This would enable a 64-processor machine to execute 64*32 = 2048 threads in parallel. Server hardware performance is set to expand rapidly for those applications that can take advantage of hyper-parallel computing.

Please replace the paragraph beginning on page 6, line 10, with the following rewritten paragraph:

An example of a dataflow graph development system is found in U.S. Patent No. 5,999,729. An example of a deadlock resolution system in a multi-threaded environment is found in U.S. Patent No. 6,088,716. Deadlock detection and correction in process networks are known, see, R. Stevens, M. Wan, P. Laramie, T. Parks & E. Lee, *Implementation of Process Networks in Java*, ~~http://www.ait.nrl.navy.mil/pgmt/ PNpaper.pdf,~~ July 1997. An example of a parallel programming environment is found in U.S. Patent No. 6,311,265. All references cited herein are incorporated by reference.

Please replace the paragraph beginning on page 7, line 15, with the following rewritten paragraph:

Broadly speaking the invention contemplates a method for developing a dataflow application where one or more data transformations is developed using a host language and several data transformations having ports are assembled into a map component with links between ports using a declarative language for static assemblage and a host language for dynamic assemblage. The host language for data transformation logic in the preferred embodiment is Java®. One or more map components are compiled with syntactic and semantic analysis and the compiled map components are synthesized into an executable dataflow application including removing the design time links between ports.

Please replace the paragraph beginning on page 12, line 12, with the following rewritten paragraph:

Data management system 10 requires, in addition to its own packages, a host language development environment (currently Java®), a third party environment required to develop the host language components as needed (in many useful cases, new map components will not require the development of host language components). Thus, the data management system's development and execution environment is a hybrid of provided tools and languages and the host language development environment.

Please replace the paragraph beginning on page 12, line 26, with the following rewritten paragraph:

The composite transformations are encapsulated into map components using a declarative, proprietary language. The atomic transformations are encapsulated into host language components (i.e. ~~Java beans~~ JavaBeans® in the current implementation). Thus, another way to think of a map in the data management system is as an assembly of map components and host language components. It should be evident that because the map is a component itself, the composition of maps is also a component. Therefore, data management system 10 supports arbitrary composition levels and map's internal structure is, in general, a tree of sub-maps.

Please replace the paragraph beginning on page 20, line 27, with the following rewritten paragraph:

Sometimes, the full interface for the map component cannot be known statically, at source creation time (for instance, a map component that reads SQL tables cannot decide the full structure of its output port only after it knows the particular table it is reading). In this case, the developer specifies all that is known about the interface at design time and leaves the rest of the specification to procedural code embedded in an interface customizer. An interface customizer is a host language component (i.e. ~~Java bean~~ JavaBean®) that implements the system provided "InterfaceCustomizer" ~~java~~ Java® interface. The procedural logic in the component is completely arbitrary and decided by the developer (in the table reader example, the interface customizer will contain logic for inspecting the table metadata and building the output port record element types accordingly). The resulting logic can be configurable since it is encapsulated in a host language component. The developer configures the interface customizer's properties using the declarative language. These properties can be configured by being set to particular values, or by delegating from interface properties of the map component.

Please replace the paragraph beginning on page 21, line 17, with the following rewritten paragraph:

There are two flavors of components that the developer can use as internal components: map processes and map components. The map process is a scalar map: it contains an executor interface (i.e. executor ports) and an atomic, natively executable, implementation. The map process is a host language component (i.e. a ~~Java Bean~~ JavaBean®) that implements the system provided class interface MapProcess. Therefore the map process implementation is not made of internal components and links but of procedural logic expressed in the host language. It is important to note that the procedural logic inside a map process does not need to contain parallel logic whatsoever. The parallelism comes when all map processes are automatically executed in parallel by executor 14. All procedural transformation work done in map components ultimately resolve to work done in a map process (for instance, the work of uppercasing a dataflow queue of strings).

Please replace the paragraph beginning on page 22, line 1, with the following rewritten paragraph:

When writing the implementation there may be cases when the developer cannot make all decisions regarding internal components and links statically. In this case, the developer puts in the source as much as she knows and then delegates the dynamic aspect to an implementation customizer. An implementation customizer is a host language component (i.e. ~~java bean~~ JavaBean®) class that implements the system provided ImplementationCustomizer ~~java~~ Java® interface. Similar to the interface customizer component, the procedural logic is completely arbitrary and decided by the developer. The resulting logic can be configurable since it is encapsulated in a host language component. The developer configures the implementation customizer's properties using the declarative language. These properties can be configured by being set to particular values, or by delegating from interface or implementation properties of the map component.

Please replace the paragraph beginning on page 23, line 10, with the following rewritten paragraph:

The target directory is used as the root of the resulting map component file location. Map components use a hierarchical name space compatible with the Java® language package notation for organization and loading of map component files.

Please replace the paragraph beginning on page 23, line 24, with the following rewritten paragraph:

The map component source includes directives for naming the package of the component and the name of the component. The fully qualified name is built from the package and the name using the same syntax as Java® fully qualified class names. The component path has the same meaning as the assembler's component path.